|  |  |  |
| --- | --- | --- |
| 南农大  **操作系统课程设计**  **实践报告**  XH2 | | |
|  | 题 目: | 可视化仿真实现作业管理与虚页内存管理 |
|  | 姓 名: | 李润隆 |
|  | 学 院: | 信息科技学院 |
|  | 专 业: | 计算机科学技术系 |
|  | 班 级: | 计科161 |
|  | 学 号: | 30216311 |
|  | 指导教师: | 姜海燕 职称: 教授 |
| 2019年3月 14 日 | | |

目录

[1.个人任务 2](#_Toc3534600)

[1.1简介 2](#_Toc3534601)

[1.2个人任务介绍 2](#_Toc3534602)

[2. 数据结构设计和仿真 4](#_Toc3534603)

[2.1页表的设计(PageList.java) 4](#_Toc3534604)

[2.2快表的设计(TLB.java) 5](#_Toc3534605)

[2.3进程控制块的设计和PCB表(PCB.java，PCBTable.java) 6](#_Toc3534606)

[2.4指令类(Instruction.java) 6](#_Toc3534607)

[2.5作业控制块JCB以及JCB队列(JCB.java，JobQueue.java) 7](#_Toc3534608)

[3.进程相关的主要操作 8](#_Toc3534609)

[3.1进程原语 8](#_Toc3534610)

[3.2进程调度 9](#_Toc3534611)

[3.3作业调度 11](#_Toc3534612)

[3.4作业以及进程信息的读入 11](#_Toc3534613)

[3.5等待队列唤醒 12](#_Toc3534614)

[4.技术分析与难点 13](#_Toc3534615)

[4.1如何设计页表类 13](#_Toc3534616)

[4.2如何将作业调度与页面调度相结合起来 13](#_Toc3534617)

[4.3如何查找等待队列里是否需要被唤醒 15](#_Toc3534618)

[4.4低级调度的指令种类处理 15](#_Toc3534619)

[5.个人总结 17](#_Toc3534620)

# 1.个人任务

## 1.1简介

我主要负责的是数据结构部分以及进程相关的操作算法等部分，主要包括数据结构设计和仿真、数据结构和硬件的共同测试、进程相关的主要操作以及最后的界面设计以及综合调试。

## 1.2个人任务介绍

1. 数据结构设计和仿真。

PageList.java

1. **public** PageList()//构造函数
2. **public** **void** show()

PCB.java

1. **public** **boolean** IsFull()//判断PCB链表是否已满
2. **public** PCB(**int** jid , **int** pid , **int** priority , **int** arrivetime , **int** intime, **int** instrnum , **int** pagenum)//构造函数

PCBTable.java

1. **public** PCBTable()

Instruction.java

1. **public** Instruction(**int** ID , **int** State , **int** NeedData , **int** TotalTime)//构造函数

JCB.java

1. **public** **void** Create()//生成作业

JCBQueue.java

1. **public** JobQueue()//初始化函数
2. **public** **boolean** IsEmpty()//判断是否为空
3. **public** **void** join(JCB e)//入队
4. **public** **void** quit()//出队
5. **public** JCB front()//队头获取
6. **public** JCB rear()//队尾获取
7. **public** **int** size()//返回大小
8. **public** **void** Show\_JCB()//显示JCB队列

TLB.java

1. **protected** **final** **static** **int** Number = 5;//假设能存储五个快表项
2. **protected** **static** **boolean** IsEmpty[];
3. **protected** **static** **int** ProID[];//进程号
4. **protected** **static** **int** PageID[];//页号
5. **protected** **static** **int** PageFrameID[];//页框号
6. **protected** **static** **int** Flag[];//标志位，记录该快表项多久未被命中
7. **protected** **static** **int** ItemNumber;//当前有多少快表项
8. 数据结构和硬件的共同测试。
9. 进程相关的主要操作。包括进程原语的编写、进程调度、作业调度过程及算法、作业及进程并发环境。

进程原语：（CPU.java）

1. **public** **void** Protect(PCB pcb)//进程的现场保护
2. **public** **void** Recover(PCB pcb)//进程的现场恢复
3. **public** **void** Run\_to\_Ready()//进程切换，运行态进程加入就绪队列，就绪队列队首进入运行态
4. **public** **void** Run\_to\_Wait()////进程切换，运行态进程加入阻塞队列，就绪队列队首进入运行态
5. **public** **void** Wake()//唤醒处于阻塞队列的
6. **public** **void** show()

进程调度、作业调度过程：

1. **class** Job\_Dispatch **extends** Thread{//作业调度线程中的低级调度部分

作业及进程并发环境：

1. **class** Job\_Dispatch **extends** Thread{//作业调度线程中的作业调度部分

作业以及进程的读入

1. **public** **void** getjob() **throws** Exception

4. 最后的界面设计以及综合调试。

Interface.java

# 数据结构设计和仿真

## 2.1页表的设计(PageList.java)

设计了一个页表类PageList和页表项类PageListItem，设定了最大页表项数目128项，用一个页表项对象数组来抽象页表，页表项类包含了页号，对应的页框号以及各种标志位，页表项类如下所示：

1. **protected** **int** PageID;//页号
2. **protected** **int** PageFrameID;//在内存中的页框号
3. **protected** **boolean** PageState;//驻留标志位，true表示当前在内存中
4. **protected** **int** PageUseState;//使用位，记录多久未被访问
5. **protected** **boolean** PageChangeState;//修改位，为true表示已被修改，需要重新写回外存
6. **protected** **boolean** IsEmpty;//判断该页表项是否为空

页表类如下所示，其中flag标志位记录了查找到的页表项的下标，在更新快表时会用到：

1. **protected** **final** **int** ItemNumber = 10;
2. **protected** PageListItem pagelistitem[];
3. **protected** **int** flag;

页表类的函数如下：

PageList()**:** 页表的构造函数对128个页表项进行了初始化，并把每个页表项的是否为空标志位设置为true。

Replace(int pageid,int pageframeid)**:**更新快表里面不常用的页表。

Delete()**：**删除已经加入到快表的项。

Insert(int pageid)**：**加入新的页表项，成功加入返回true，超出页表项上限则返回false。

Visit(int pageid)**:**查找页表，如果页面不在内存中，就发出缺页异常，否则返回对应的页框号。

## 2.2快表的设计(TLB.java)

TLB包含了如下成员，自定义快表最大能存储5个快表项，IsEmpty标志位判断该快表项是否为空，ItemNumber记录当前已经有多少个快表项，oldpageid标志长时间未被访问的项目的页号，将被替换出去

1. **protected** **final** **static** **int** Number = 5;//假设能存储五个快表项
2. **protected** **static** **boolean** IsEmpty[];
3. **protected** **static** **int** ProID[];//进程号
4. **protected** **static** **int** PageID[];//页号
5. **protected** **static** **int** PageFrameID[];//页框号
6. **protected** **static** **int** Flag[];//标志位，记录该快表项多久未被命中
7. **protected** **static** **int** ItemNumber;//当前有多少快表项
8. **protected** **int** oldproid;//旧的进程
9. **protected** **int** oldpageid;//旧的页号
10. **protected** **int** oldpageframeid;//旧的页框号

包含的函数如下：

1. **public** TLB()//构造函数
2. **public** **int** VisitTLB(**int** proid , **int** pageid)//访问快表
3. **public** **boolean** AlterTLB(**int** proid , **int** pageid , **int** pageframeid)//更新快表
4. **public** **int** Calculate()//返回最久未被访问的快表项的下标

## 2.3进程控制块的设计和PCB表(PCB.java，PCBTable.java)

创建了PCB类来表示PCB相关的信息，存储的主要变量有：

1. **protected** **int** Job\_ID;//进程所属的作业ID
2. **protected** **int** Pro\_ID;//进程ID
3. **protected** **int** Pro\_Priority;//进程优先级
4. **private** **int** Pro\_ArriveTime;//进程到达时间
5. **private** **int** Pro\_InTime;//进程创建时间
6. **private** **int** Pro\_State;//进程状态，1运行，2就绪，3等待
7. **protected** **int** Pro\_RunTime;//进程运行时间
8. **protected** **int** Pro\_EndTime;//进程结束时间
9. **private** **int** Pro\_TotalTime;//进程周转时间
10. **protected** **int** PSW;//进程当前执行的指令编号
11. **protected** **int** Pro\_InstrNum;//进程包含的指令数目
12. **private** Instruction[] instruction;//指令链表

创建PCBTable类管理各个队列

1. **protected** PCB RunQueue;//指向正在运行的进程
2. **protected** MyQueue ReadyQueue;//就绪队列
3. **protected** MyQueue WaitQueue;//等待队列
4. **protected** MyQueue FinishQueue;//已完成队列

以及一些进程原语的实现：进程的初始化、进程的创建、进程的终止、进程的阻塞、进程的唤醒。  
进程的创建：申请空白PCB，为新进程分配资源，初始化PCB，将PCB插入就绪队列。  
进程的终止：根据进程标识符读取PCB，释放全部资源，将PCB从队列移除。  
进程阻塞：停止本进程，改变PCB，调度新进程。  
进程唤醒：将PCB从阻塞队列移除，更新PCB，将PCB插入到就绪队列。

## 2.4指令类(Instruction.java)

指令类仿真：仿真指令的数据结构，包含了指令ID、指令状态标志、单条指令运行时间、指令地址。这些都是每条指令的基本组成项，具体如下：

1. **protected** **int** Instr\_ID;//指令序号
2. **protected** **int** Instr\_State;//指令类型，0表示系统调用，1表示用户态计算操作，2表示PV操作,3表示IO操作
3. **protected** **int** Need\_Data;//指令是否需要访问数据段，1表示需要，0表示不需要
4. **protected** **int** Instr\_TotalTime;//指令总时间
5. **protected** **int** Instr\_RunTime;//指令已经运行的时间
6. **protected** **int** Instr\_Addr;//指令的地址

本试验假设：

系统调用指令：在执行一条机器指令时必须完成，不可以中断。

Instr\_State：0表示系统调用指令、1表示用户态计算操作指令、2表示PV操作指令，3表示I/O 操作，用随机数产生0、1、2、3。

Instr\_RunTime：该条指令完成需要的时间， 随机函数产生[20-50]之间的随机10ms倍数的整数；当State=3时，用户进程发出I/O阻塞请求，并假设完成

I/O数据通信时间为Instr\_RunTime，之后可唤醒。

Instr\_Addr：逻辑地址：页号+页内偏移，每条指令的逻辑地址随机生成

## 2.5作业控制块JCB以及JCB队列(JCB.java，JobQueue.java)

**JCB：**

为每个进入系统的作业建立JCB，包含作业号，作业到达时间，作业包含的进程数。pro[][]数组存放每个进程的信息。pro\_position[]存放每个进程每个页面的外存位置，以下为主要变量：

1. **protected** **int** JCB\_ID;//作业号
2. **protected** **int** InTime;//作业到达时间
3. **protected** **int** PRO\_Number;//作业包含的进程数
4. **protected** JCB next;//指向下一个作业节点
5. **protected** **int** pro[][];//存放随机生成的进程的详细信息
6. **protected** **int** pro\_position[];//存放每个进程每个页面的外存位置

以下为主要功能函数：

1. **public** **void** Create()//生成作业
2. **public** **void** WriteOutPageList() **throws** FileNotFoundException//往磁盘里写外页表，每个进程一张外页表
3. **public** **void** WritePage(**int** a,**int** x,**int** y,**int** k)//往磁盘里写每个页面的详细信息

**JobQueue：**

JobQueue类仿真作业队列，包含了入队，出队，显示队列，返回作业数目等函数，详细变量如下：

1. **protected** JCB front;//队头
2. **protected** JCB rear;//队尾
3. **protected** **int** Queue\_Size;

使用了一个链队列来保存作业队列，该类保存了队列相关的操作。

1. **public** JobQueue()//初始化函数
2. **public** **boolean** IsEmpty()//判断是否为空
3. **public** **void** join(JCB e)//入队
4. **public** **void** quit()//出队
5. **public** JCB front()//队头获取
6. **public** JCB rear()//队尾获取
7. **public** **int** size()//返回大小
8. **public** **void** Show\_JCB()//显示JCB队列

# 3.进程相关的主要操作

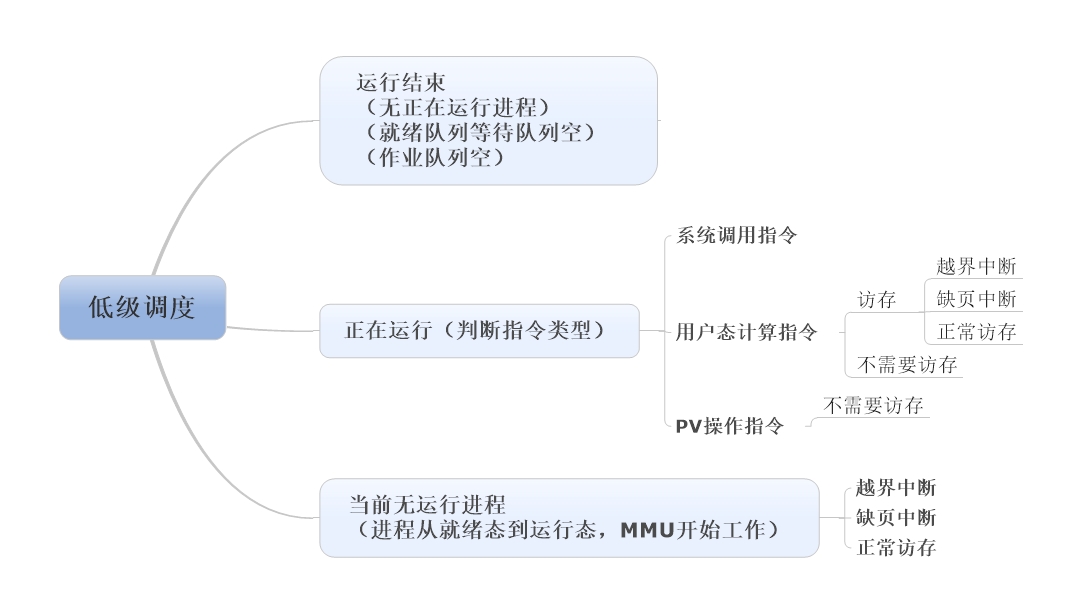
包括进程原语的编写、进程调度、作业调度过程及算法、作业及进程并发环境。

## 3.1进程原语

1. **public** **void** Protect(PCB pcb)//进程的现场保护（返回相应的寄存器数据）
2. **public** **void** Recover(PCB pcb)//进程的现场恢复 （保存相应的寄存器数据）
3. **public** **void** Run\_to\_Ready()//进程切换，运行态进程加入就绪队列，就绪队列队首进入运行态
4. **public** **void** Run\_to\_Wait()////进程切换，运行态进程加入阻塞队列，就绪队列队首进入运行态
5. **public** **void** show()

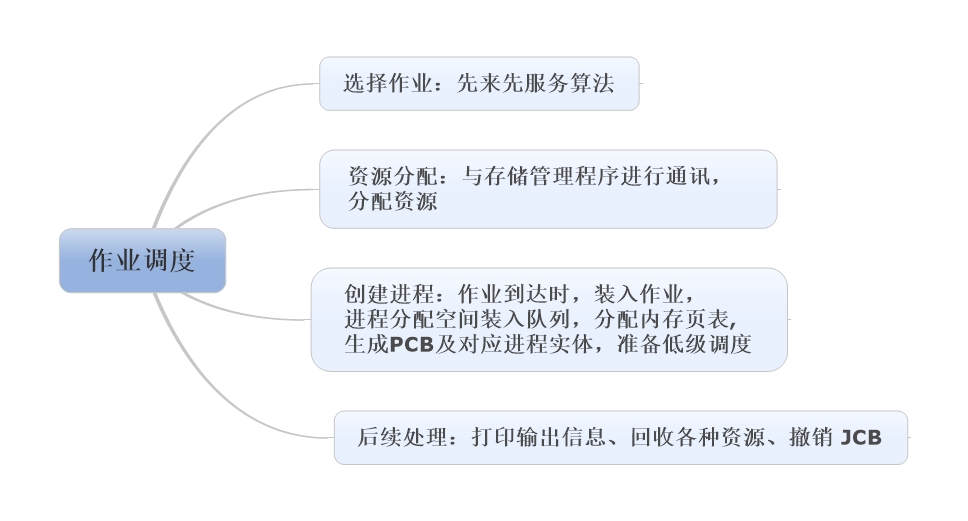
进程切换的时候主要通过现场保护，相应队列的入队以及出队来实现进程切换，并且访问MMU。

## 3.2进程调度



1. //低级调度
2. **if**(无正在运行进程，就绪等待进程队列空，作业队列空)
3. 输出结束信息
4. **if**(如果当前有进程在运行)
5. {分析指令类型
6. 系统调用指令：
7. 指令运行10ms
8. **if**(当前指令运行完毕）
9. {PC自增，指向该进程下一条指令;
10. **if**(该进程所有指令全部运行完毕）
11. {将该进程加入完成队列
12. 回收资源
13. 设置空值
14. }**else** {
15. 放入就绪态
16. }
17. }
18. **break**;
19. 用户态计算指令：
20. **if**(访存）
21. {寻找位置
22. 循环寻找所有页面
23. {MMU开始工作
24. **switch** (pageframeid){
25. **case** -1:
26. 越界中断处理
27. **break**;
28. **case** -2:
29. 缺页中断处理，将缺页调入内存
30. **break**;
31. **default**:
32. **break**;
33. }
34. }
35. }
36. 指令运行10ms
37. **if**(当前指令运行完）
38. PC++;//PC自增，指向该进程下一条指令;
39. **if**（该进程所有指令全部运行完毕）
40. 进程结束放入完成队列回收资源
41. **else** **if**(时间片到）
42. 进入就绪态
43. **break**;
44. PV操作指令：
45. **if**（该进程需要访问数据）
46. 找到位置
47. **for**遍历需要的页面
48. MMU开始工作
49. **switch** (pageframeid)
50. **case** -1:
51. 越界中断处理
52. **case** -2:
53. 缺页中断处理，将缺页调入内存
54. **default**:
55. **break**;
56. **if**(临界区可以访问)
57. 指令运行10ms
58. **if**(当前指令运行完毕）
59. 地址自增，其他进程可以访问临界资源
60. **if**(该进程所有指令全部运行完毕）
61. 进程完成操作
62. **else**
63. 运行转就绪
64. **break**;
65. **default**:
66. **break**;
67. }
68. IO操作指令
69. 记录进程状态，更新等待标志位，加入等待队列
70. 运行指针置为空
71. 如果就绪队列非空
72. 就绪态转为运行态
73. }**else**
74. 如果当前没有进程在运行
75. 进程进入运行相关操作
76. 输出信息

## 3.3作业调度



作业调度Job\_Dispatch单独成为一个线程，通过加锁来解决与时钟线程的并行问题，作业调度算法使用的是先来先服务算法。代码分析如下：

若该时刻有作业并且作业到达（这里是作业调度线程的主要部分，使用了先来先服务的算法）

到达作业数++

分配每个进程（与存储部分进行通讯）

创建PCB

创建该作业的每一个进程并将进程加入就绪队列

为新进程分配页表

为每个进程的每个页分配内存并且更新该进程的页表

将已经调度成进程的作业出队，撤销JCB

## 3.4作业以及进程信息的读入

**public** **void** getjob() **throws** Exception//从文件中读取作业信息

{

打开作业数目记录文件

读取开头部分本次调度的作业数

读取每一个作业的详细信息的循环

{

打开作业详情文件

读取作业号

读取作业到达时间

创建jcb节点

读取该作业的进程数

jcb存放每个进程的详细信息

jcb存放每个进程外页表的位置

获取每个进程的详细信息

加入作业队列

}

}

## 3.5等待队列唤醒

如果等待队列非空，寻找可以唤醒的进程

{

获取等待队列里节点的个数

创建pcb数组保存不需要等待的进程

创立指针指向第一个pcb

从头到尾检查每一个节点是否可以被唤醒

{

如果该进程因I/O操作中断且等待事件结束

{

运行完该条I/O指令

判断是否同步完成

修改该进程的等待标志位为没有等待

更新进程状态为就绪态

更新进程已运行的指令数

加入就绪队列

如果有被同步的进程且同步该进程的进程已经运行完200条指令

处理同步结束操作

else无法被唤醒，也就不需要出队

保存不需要出队的进程

}

检查下一个节点

}

由于该需要等待的已经保存到了数组当中，等待时间结束的都已经出队，因此可以清空等待队列将还需要等待的PCB放入

重置等待队列

将无法被唤醒的进程加入等待队列

}

}

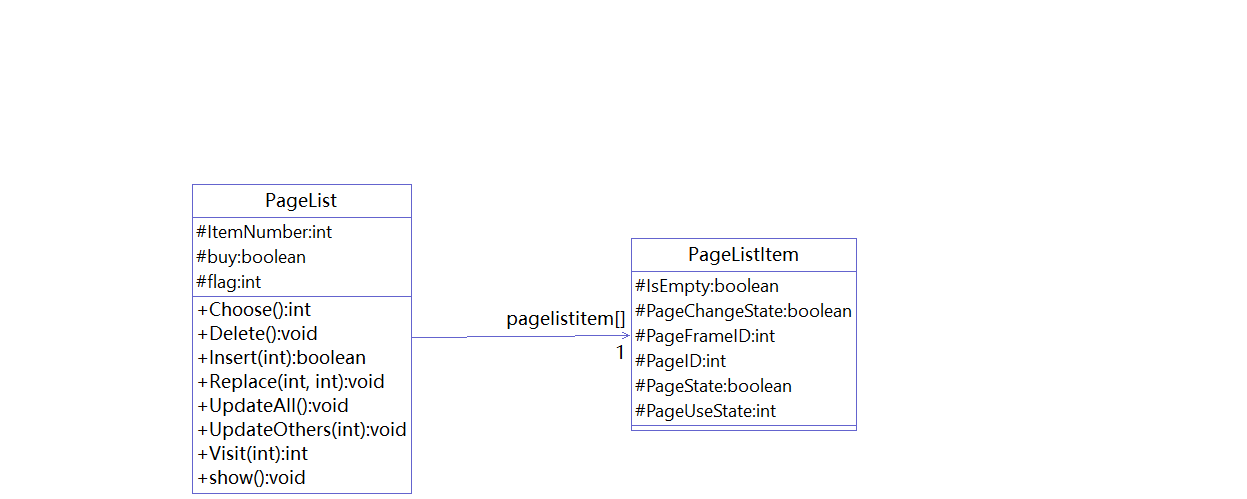
这里通过数组保存仍然需要等待的进程控制块，在等待结束进程出队以后再放入。

# 4.技术分析与难点

## 4.1如何设计页表类

问题描述：页表作为一个重定位寄存器的集合，需要被多次访问，如何设计这一个特殊的结构？

问题解决：

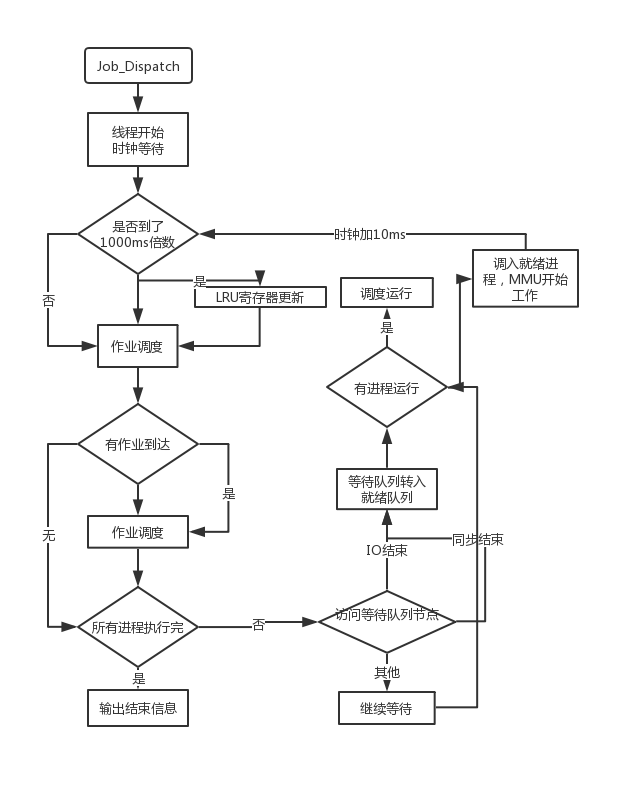


设计了两个类分别是用来表示页表的Pagelist以及页表项的PageListltem，通过实例化的方式来使得后者被包含。

## 4.2如何将作业调度与页面调度相结合起来

问题描述：上学期的课外必修实验没有要求到页面调度，但是这一次需要将指令存储位置，访问资源等都考虑进来。

问题解决：在获取作业的时候需要为每一个进程创建PCB节点，以及自动分配资源，通过MMU来分配得到页表，并且为每一个页都分配内存。流程图如下：



伪码描述如下：

1. **若该时刻有作业并且作业到达（这里是作业调度线程的主要部分，使用了先来先服务的算法）**
2. **到达作业数++**
3. **分配每个进程（与存储部分进行通讯）**
4. **创建PCB**
5. **创建该作业的每一个进程并将进程加入就绪队列**
6. **为新进程分配页表**
7. **为每个进程的每个页分配内存并且更新该进程的页表**
8. **将已经调度成进程的作业出队，撤销JCB**

## 4.3如何查找等待队列里是否需要被唤醒

问题描述：等待队列里面不仅有因为IO操作导致的等待，还有进程同步导致的等待，等待队列里面可能有多个PCB需要出队。

问题解决：使用了一个PCB数组来顺序保存仍然需要等待的PCB块，通过指针遍历来检查每个等待的进程控制块的等待事件是否结束或者是同步事件有没有结束，如果结束了就转为就绪态，否则就保存在PCB数组中等结束的时候再放回去，这样可以保持等待队列的顺序不变并且将等待结束的进程进行全部出队。

## 4.4低级调度的指令种类处理

问题描述：由于设计了四种指令，系统调用指令，用户态计算指令，PV操作指令，I/O操作指令，各个指令对于是否访问数据段，是否发生等待事件，是否一次运行完毕还是通过时间片轮转算法来调度运行都有着不同的要求。

问题解决：通过判断指令种类的标志Instr\_State来分别处理不同的指令，并且根据是否需要访问数据段选择使用MMU。伪码如下:

1. //低级调度
2. **if**(无正在运行进程，就绪等待进程队列空，作业队列空)
3. 输出结束信息
4. **if**(如果当前有进程在运行)
5. {分析指令类型
6. 系统调用指令：
7. 指令运行10ms
8. **if**(当前指令运行完毕）
9. {PC自增，指向该进程下一条指令;
10. **if**(该进程所有指令全部运行完毕）
11. {将该进程加入完成队列
12. 回收资源
13. 设置空值
14. }**else** {
15. 放入就绪态
16. 用户态计算指令：
17. **if**(访存）
18. {寻找位置
19. 循环寻找所有页面
20. {MMU开始工作
21. **switch** (pageframeid){
22. **case** -1:越界中断处理
23. **break**;
24. **case** -2: 缺页中断处理，将缺页调入内存
25. 指令运行10ms
26. **if**(当前指令运行完）
27. PC++;//PC自增，指向该进程下一条指令;
28. **if**（该进程所有指令全部运行完毕）
29. 进程结束放入完成队列回收资源
30. **else** **if**(时间片到）
31. 进入就绪态
32. **break**;
33. PV操作指令：
34. **if**（该进程需要访问数据）
35. 找到位置
36. **for**遍历需要的页面
37. MMU开始工作
38. **switch** (pageframeid)
39. **case** -1:
40. 越界中断处理
41. **case** -2:
42. 缺页中断处理，将缺页调入内存
43. **if**(临界区可以访问)
44. 指令运行10ms
45. **if**(当前指令运行完毕）
46. 地址自增，其他进程可以访问临界资源
47. **if**(该进程所有指令全部运行完毕）
48. 进程完成操作
49. **else**
50. 运行转就绪
51. **break**;
52. **default**:
53. **break**;
54. }
55. IO操作指令
56. 记录进程状态，更新等待标志位，加入等待队列
57. 运行指针置为空
58. 如果就绪队列非空
59. 就绪态转为运行态
60. }**else**
61. 如果当前没有进程在运行
62. 进程进入运行相关操作
63. 输出信息

# 5.个人总结

过去的一个多学期过的比较充实，特别是寒假开学的三周以来，通过学习操作系统书并且将其中章节付诸实践，提高了对处理器管理的理解，包括中断机制，进程的基本概念以及实现，以及处理器的调度算法。同时在使用java语言的过程中，提高了对于java多线程以及界面的设计能力。

通过查阅源码等资料对Linux操作系统有了更加深入的认识，对于内存管理、进程管理以及虚存管理有了进一步的了解，尤其是理论考试中不是特别掌握的部分比如缺页的具体处理也进行了学习，并且在上一学期的必修实验基础上对于进程结构继续细化，分层次的设计了仿真实现内存管理以及虚存管理机制。个人主要完成了数据结构部分以及进程相关的操作算法等部分，主要包括数据结构设计和仿真、数据结构和硬件的共同测试、进程相关的主要操作以及最后的界面设计以及综合调试。同时，在和队友的共同测试，相互之间的协作以及后期的共同优化修改中，也学到了一些沟通方面的经验。第一次团队协作完成一个较大的具有整体性系统性地操作系统的一部分，是对自己的一次考验，也是对操作系统理论知识的巩固和增强。